Manual

Italo Ivo Pinto

August 2022

1 Functions

dynamic_louvain(adj_matrix, gamma=1, omega=1, ci=None, seed=None)

Calculates the time evolving community structure using the generalized Louvain algorithm (see Jutla et al. 2011), using gamma as a spatial resolution parameter and omega as temporal resolution parameter.

Parameters

adj_matrix : The adjacency matrix from which the algorithm will generate the community structure.
gamma : Spatial resolution parameter.
omega : Temporal resolution parameter.
ci : Initial community structure, if not specified or None will start with each node in a different community.
seed : Seed for the random number generator.

Returns

ci : Final community structure generated, it is stored in a numpy array with shape (time,nodes).
gamma : Spatial resolution value used.
omega : Temporal resolution value used.
q : Quality metric value of the final community structure.

parameter_search(matrices, lower_gamma, higher_gamma, lower_omega, higher_omega, iterations, num_processors = 10, points=10):

Searches for the gamma and omega parameters that minimizes the spatial skewness and maximizes the scalefreeness for the adjacency matrices used.

Parameters

matrices : Adjacency matrices to use to obtain the community structures.
lower_gamma : Minimum value of gamma to consider in the parameter search.
higher_gamma : Maximum value of gamma to consider in the parameter search.
lower_omega : Minimum value of omega to consider in the parameter search.
higher_omega : Maximum value of omega to consider in the parameter search.
iterations : Number of iterations to perform.
num_processors : Number of threads used to calculate the data points in each iteration, if not specified the default value of 10 is used.
points : Number of data points used to calculate skewness and scalefreeness between the lower and higher values of gamma and omega for each iteration, if not specified the default value of 10 is used.

Returns

gamma_omega_sequence : Values of optimized gamma and omega for each iteration.
step_comm_sizes : Values of gamma and absolute value of skewness for the values used in each iteration.
step_durations : Values of omega and mean distance between the cumulative distributions of the model and the data for each iteration.

comm_time_calc(matrix)

Calculates the distribution of time nodes take to change community.

Parameters

matrix : Adjacency matrix.

Returns

comm_times : Distribution of time a node takes to change community.

cumulative_distances_calc(data,lower_threshold)

Calculates the distance between the curves of cumulative distributions of the data and the values generated by a power-law model with exponent calculated by the maximum likelihood estimator method.

Parameters

data : Distribution of values to fit the model.
lower_threshold : minimum value to consider for the comparison

Returns

ks_distances: Mean of the distances between the curves, the mean is calculated over the existing bins.

mle(x, xmin=None, xmax=None)

Calculates the exponent of a power law using the maximum likelihood estimator method.

Parameters

x : Distribution of values to fit the model.
xmin : minimum value to consider for fitting the power law
xmax : maximum value to consider for fitting the power law

Returns

tau : Estimated exponent calculated.
Ln : Log-likelihood for the data to be sampled from the model obtained.

synt_data_generator(a,comm_sizes,lower_threshold):

Generates a synthetic series of values using the power law model calculated using MLE.

Parameters

a : power law exponent
comm_sizes : Distribution of values used to fit the model.
lower_threshold : minimum value used for fitting the power law.

Returns

synt_data : synthetic data obtained from the model within the same range as the original data.

comm_size_calc(matrix)

Calculates the community size distribution for all time layers.

Parameters

matrix : dynamic community structure.

Returns

comm_sizes : list with the community sizes in the dynamic community structure.

2 Dynamic community metrics

In this section we describe the functions to calculate dynamic community structure metrics, for more information on details of the metrics described below we refer to Garcia et al. 2018.

flexibility_calc(comm_structure)

Calculates the flexibility metric for a community structure.

Parameters

comm_structure : dynamic community structure as a numpy array with shape (time,nodes).

Returns

flexibility : Numpy array with the flexibility value for each network node.

disjointedness_calc(comm_structure)

Calculates the disjointedness metric for a community structure.

Parameters

comm_structure : dynamic community structure as a numpy array with shape (time,nodes).

Returns

disjointedness : Numpy array with the flexibility value for each network node.

cohesion_calc(comm_structure)

Calculates the cohesion matrix and cohesion strength for a community structure.

Parameters

comm_structure : dynamic community structure as a numpy array with shape (time,nodes).

Returns

cohesion_matrix : A 2-d numpy array with shape (nodes,nodes) and the pairwise cohesion for each node pair.
cohesion_str : A numpy array with the cohesion strength for each node of the network.

promiscuity_calc(comm_structure)

Calculates the promiscuity for a community structure.

Parameters

comm_structure : dynamic community structure as a numpy array with shape (time,nodes).

Returns

promiscuity : A numpy array with the promiscuity value for each node.

allegiance_calc(comm_structure)

Calculates the pairwise allegiance of a community structure.

Parameters

comm_structure : dynamic community structure as a numpy array with shape (time,nodes).

Returns

allegiance : matrix which stores the pairwise allegiances averaged on time layers.

intermittence_calc(comm_structure)

Calculates the pairwise intermittence of a community structure.

Parameters

comm_structure : dynamic community structure as a numpy array with shape (time,nodes).

Returns

intermittence : matrix which stores the pairwise intermittence averaged on time layers.